Minebea

PCI インタフェースボード

OPFT-SCPCI

取扱説明書

ミネベア株式会社
計測機器事業部

294-1215-D

はじめに

このたびは、ミネベア製 PCI インタフェースボード OPFT-SCPCI をご採用いただき、誠にありがとうございます。本取扱説明書は、ご使用いただく場合の取り扱い、留意点について説明しています。誤った取り扱いは思わぬ不具合を引きおこしますので、ご使用前に必ず本取扱説明書を一読され、正しくご使用いただきますようお願いいたします。

本取扱説明書につきましては、ご使用になるお客様のお手元に届くようご配慮をお願いいたします。

本書で使用しているマークと約束事について

本書では絶対にしないでいただきたいことや、注意していただきたいこと、及び参考にしていただきたいことの説明には、次のようなマークを付けています。

これらのマークの箇所は必ずお読み下さい。

⚠ 警告

取扱を誤った場合に使用者が死亡、又は重傷を負う危険性が想定される場合についての記述です。
ここに説明されているようなことは絶対に行わないで下さい。

⚠ 注意

取扱を誤った場合に使用者が傷害を負う危険が想定される場合、及び物的損害のみの発生が想定される場合についての記述です。

操作や作業する上での注意や制限などです。
誤動作を防止する為に、必ずお読み下さい。

- 本書に記載されている内容は、改良の為予告なく変更される場合がありますので、あらかじめご了承下さい。
- Microsoft Windows 2000、Microsoft Visual Basic、Microsoft Visual C++は、米国 Microsoft Corporation の各国における登録商標または、商標です。その他、本文中で使用している商品名は、各社の商標または登録商標です。

安全にお使いいただくために

ご使用になる前に、本文を必ずお読み下さい。

1．設置場所について

⚠ 注意

温度、湿度が以下の範囲内の場所でご使用下さい。

- 周囲温度：0 ℃ to 50 ℃
- 周囲湿度：20 %RH to 90 %RH（結露のなきこと）

⚠ 警告

本器を次のような場所に設置しないで下さい。思わぬ故障の原因になることがあります。

① 設置してはいけない場所

- ほこりや粉塵が多い雰囲気での使用は避けて下さい。
- 腐食性ガス、塩分等を含んだ雰囲気での使用は避けて下さい。

⚠ 注意

本ボードには多数の電気部品が搭載されていますが、これらは静電破壊を起こす恐れがあります。
取扱い前には本ボード以外の金属部等に触れ、作業者自身の放電を行って下さい。又、取扱い時も電気部品には触れぬ様にご注意下さい。

② 本器を実装するとき

- ボード上には、あらかじめ設定が必要なジャンパがあります。PCI バススロットに実装する前に必ず確認して下さい。
- ボード上のジャンパは、指定以外の設定にしないで下さい。誤動作、故障の原因となります。
- ボードに衝撃を与えたり、曲げたりしないで下さい。誤動作、故障の原因になります。
- ボードの金メッキ端子部（エッジコネクタ）には、手を触れないで下さい。
  誤動作、故障の原因となります。
- パソコンの電源が入った状態で、ボードを PCI バススロットに実装したり、抜いたりしないで下さい。誤動作、故障の原因となります。必ずパソコンの電源を切ってから実施して下さい。
- パソコンから、実装するボードに十分な電力が供給できる事を確認して下さい。十分な電力が供給されない場合は、誤動作、故障の原因となります。

改訂履歴

日付	取扱説明書　No.	改訂理由（内容）
2004/09	DRW. NO.294-1215	初版
2005/04	DRW. NO.294-1215-A	FN05-02020 による ー誤記訂正ー 全体的に英数字のフォント、及び表現方法の見直し
2006/06	DRW.NO294-1215-B	FN06-02088 による ー改訂ー ・Linux に関する記述追記 ・付属品フロッピーディスクから、CD-ROM に変更
2009/08	DRW.NO294-1215-C	FN09-02096 による ー改訂ー ・質量の記載改定
2010/08	DRW.NO294-1215-D	FN10-02126 による ー改訂ー ・ミネベアロゴの変更

目　　次

はじめに ..... *I*

本書で使用しているマークと約束事について ..... *I*

安全にお使いいただくために ..... *II*

1. 設置場所について II

改訂履歴 ..... *III*

1. はじめに *1*

1-1. 概要 1
1-2. 特徴 1
1-3. 動作環境 1
1-4. 付属品 1
1-5. ボード外形 2

2. セットアップ *3*

2-1. ボード本体コネクタ、ジャンパーの設定 3
2-2. ボードの実装 4
2-3. ハードウェアのインストール(WINDOWS 版　ドライバ) 5
2-4. 接続 7

3. 機能の説明(WINDOWS 対応ドライバ) *8*

3-1. 付属ドライバ CD-ROM 内容 8
3-2. アプリケーション開発用ファイル使用方法 8
3-3. 関数説明 9
3-4. 各関数の使用シーケンス例 15

4. 機能の説明(LINUX 対応ドライバ) *16*

4-1. 付属ドライバ CD-ROM 内容 16
4-2. 関数説明 16
4-3. RT LINUX API 一覧 23
4-4. IOCTL 一覧 24
4-5. アプリケーション　サンプルプログラム 26
4-6. デバイスファイルの作成(リアルタイム LINUX では無い場合に必須) 32
4-7. センサ・OPFT-SCPCI 基板のやり取りの流れ 33
4-8. コンパイル、ドライバ組込み手順 34

5. ドライバのアンインストール *37*

5-1. ドライバのアンインストール(WINDOWS 版　ドライバ) 37
5-2. ボードの取り外し(WINDOWS 版、LINUX 版　ドライバ共通) 37

6. 仕様 *38*

6-1. 基本仕様 38

7. 保証 39
7-1. 保証 39
7-2. 修理 39

１．はじめに

１-１． 概要

このボードは、弊社製6軸センサのデータ取込みを行う為のPCIバス準拠のインタフェースボードです。
付属 API を組み込んだアプリケーションを作成する事により、ユーザーはセンサデータの取込、加工等が自由に行えます。

１-２． 特徴

- センサに電力を供給します。
- ボード 1 枚で、センサを 4 個まで接続できます。
- ボードは、同時に 2 枚まで使用可能です。
- ボードとセンサの間は、RS-422 通信で行います。
- センサからのデータは、最大で 10 000 回/秒取込が行えます。
- センサからの出力（シリアル通信）を、ドライバが確保したパソコン内メモリにバッファリングするので、アプリケーションはバッファリングしたデータを取込む事が出来ます。
- Windows 用と Linux 用ドライバソフトを同梱していますので、ご使用の OS に合わせてアプリケーションソフトを作成する事が可能です。

１-３． 動作環境

（１）OS(オペレーティングシステム)

①Windows 環境

Microsoft Windows 2000 Professional SP3 以上

②Linux 環境

カーネル 2.4.X、あるいは　カーネル 2.6.X

(２)　メモリー

256 Mbyte 以上

（３）CPU

Pentium Ⅲ 800 MHz 以上(シングル CPU)

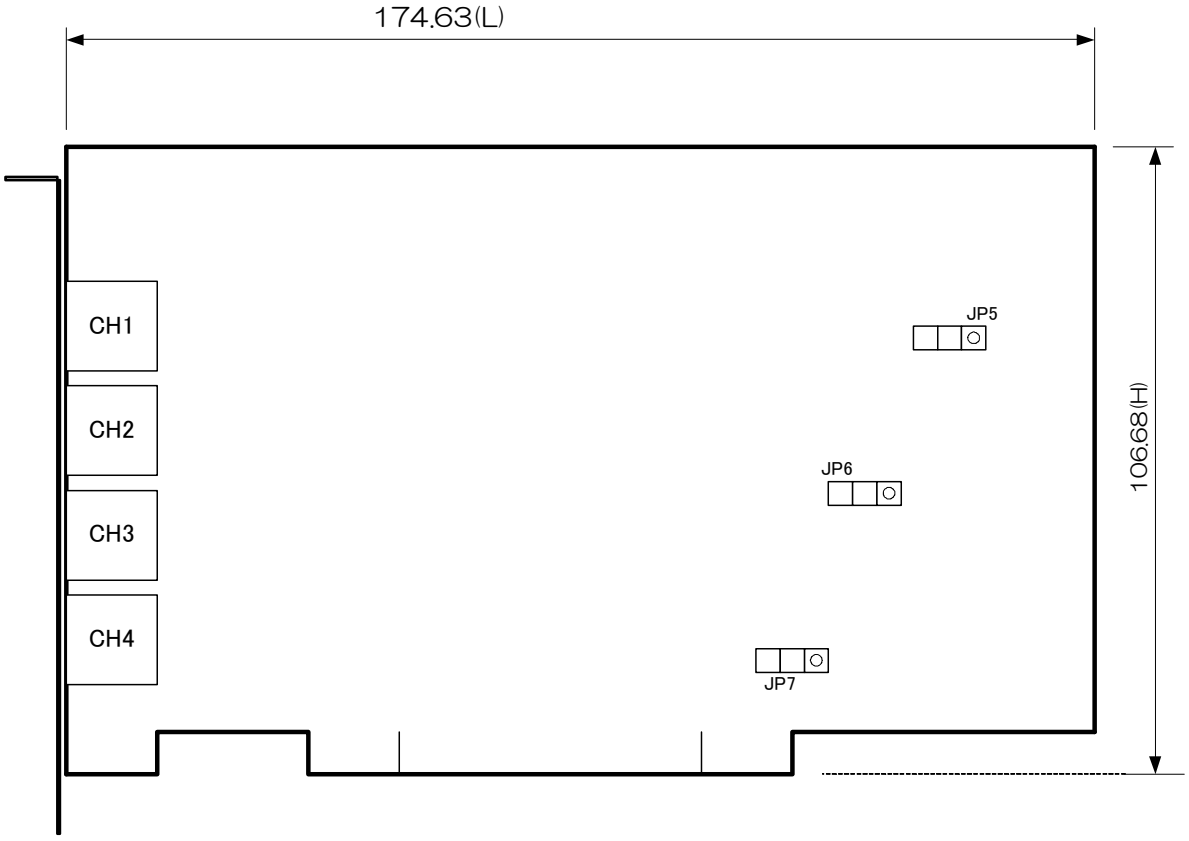
１-４． 付属品

(１)　ドライバ CD-ROM …… 1 枚

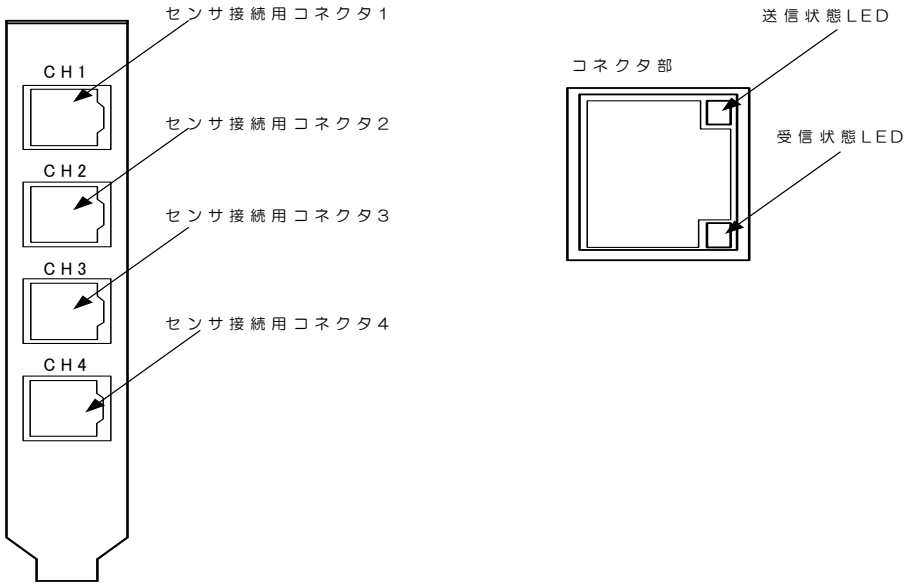
(２)　取扱説明書 …… 1 冊(本書)

1-5. ボード外形

ボードの外形寸法



ボードパネル面図



※センサとの通信が行われている時は、コネクタ部の LED が点灯します。

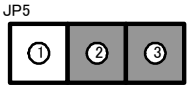
2．セットアップ

2-1． ボード本体コネクタ、ジャンパーの設定

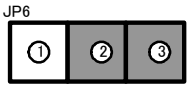
本ボードには、JP5、JP6、JP7 の 3 個のジャンパーがあります。

本ボードを、パソコンに実装する前にジャンパーの設定を行って下さい。本ボードを 1 枚のみ使用時は、工場出荷状態でそのままご使用できます。

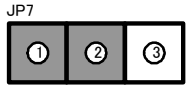
（1） ジャンパ工場出荷時設定



チャネル0～3での使用を設定



本ボードでは、２－３ショート固定で使用します。
※絶対に、JP6 設定は変更しないで下さい。



本ボードでは、１－２ショート固定で使用します。
※絶対に、JP7設定は変更しないで下さい。

（2） JP5 設定とコネクタ・チャネル対応表

OPFT-SCPCI		1 枚使用時	2 枚使用時	
JP5 状態	コネクタ	CH 番号	CH 番号	
2-3 ショート	CH1	0	0	1 枚目ボード CH1 に接続したセンサのチャネル番号
	CH2	1	1	1 枚目ボード CH2 に接続したセンサのチャネル番号
	CH3	2	2	1 枚目ボード CH3 に接続したセンサのチャネル番号
	CH4	3	3	1 枚目ボード CH4 に接続したセンサのチャネル番号
1-2 ショート	CH1	---	4	2 枚目ボード CH1 に接続したセンサのチャネル番号
	CH2	---	5	2 枚目ボード CH2 に接続したセンサのチャネル番号
	CH3	---	6	2 枚目ボード CH3 に接続したセンサのチャネル番号
	CH4	---	7	2 枚目ボード CH4 に接続したセンサのチャネル番号

※本表は、アプリケーションソフトで指定するチャネル番号と、ボードのコネクタとの関係を示しています。

2-2.ボードの実装

(1)　ボードを実装する前に、システムをシャットダウンし、コンセントからパソコンの電源ケーブルを抜いてください。

(2)　パソコン本体からケーブル類を外し、カバーを取り外して下さい。

(3)　空いている PCI バススロットのスロットカバーを取り外します。
取り外したネジは、本ボードの固定に使用します。
(PCI バススロット位置は、パソコン本体のマニュアルで確認して下さい)

(4)　本ボードを、PCI バススロットに実装してください。

(5)　ボードのブラケットを、手順(3)で取り外したネジで固定します。

(6)　パソコンのカバーを取り付け、ケーブル類を元の状態にして下さい。

⚠ 注意

● ボードに衝撃を与えたり、曲げたりしないで下さい。誤動作、故障の原因になります。

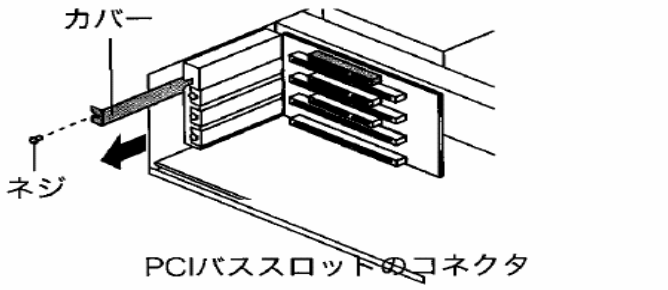
本ボードには多数の電気部品が搭載されていますが、これらは静電破壊を起こす恐れがあります。
取扱い前には本ボード以外の金属部等に触れ、作業者自身の放電を行って下さい。又、取扱い時も電気部品には触れぬ様にご注意下さい。

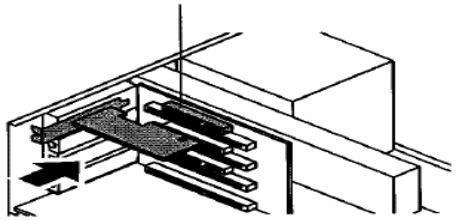
● ボードの金メッキ端子部（エッジコネクタ）には、手を触れないで下さい。
誤動作、故障の原因となります。

● パソコンの電源が入った状態で、ボードを PCI バススロットに実装したり、抜いたりしないで下さい。誤動作、故障の原因となります。必ずパソコンの電源を切ってから実施して下さい。

⚠ 注意

本ボードが確実に、PCI バススロットに差込まれている事（エッジコネクタが見えなくなるまで差込まれている、PCI バススロットに対して水平である）を確認して下さい。確実に差込まれていないと、パソコンの電源 ON 時に BIOS が、本ボードを認識できなくなり BIOS がエラー表示を行う事があります。





2-3. ハードウェアのインストール(Windows 版　ドライバ)

※OS が、Windows の場合は　2-3 項手順のハードウェアのインストールが必要です。

2-3-1.パソコンの電源投入

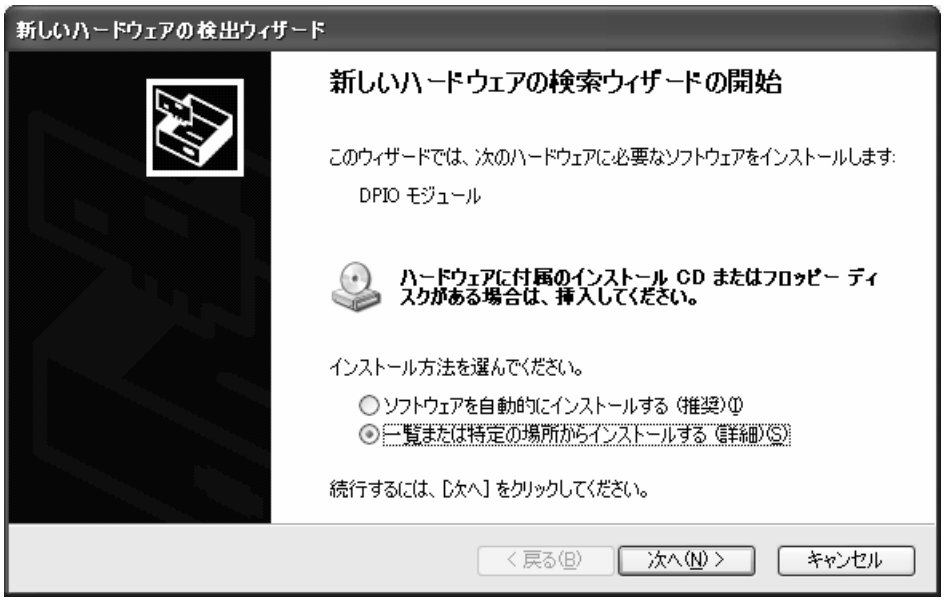
パソコンの電源を入れて下さい。

コンピュータの管理者権限があるユーザー（Administator 等）でログインして下さい。

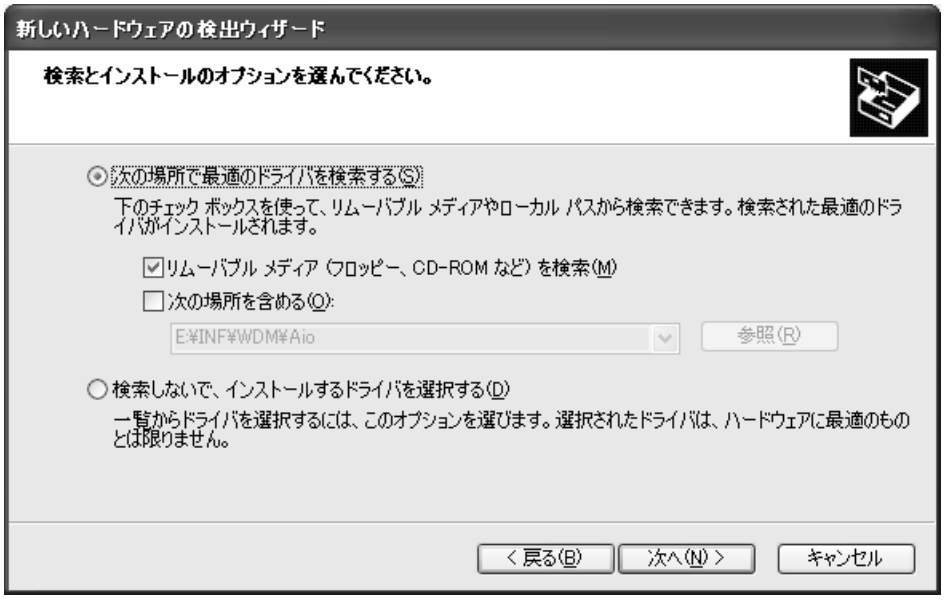
2-3-2.新しいハードウェアの追加

(1)　「新しいハードウェアの検出ウィザード」が起動します。

[一覧または特定の場所からインストールする]を選択し、「次へ(N)」ボタンをクリックして下さい。



(2)　次の画面が表示されます。

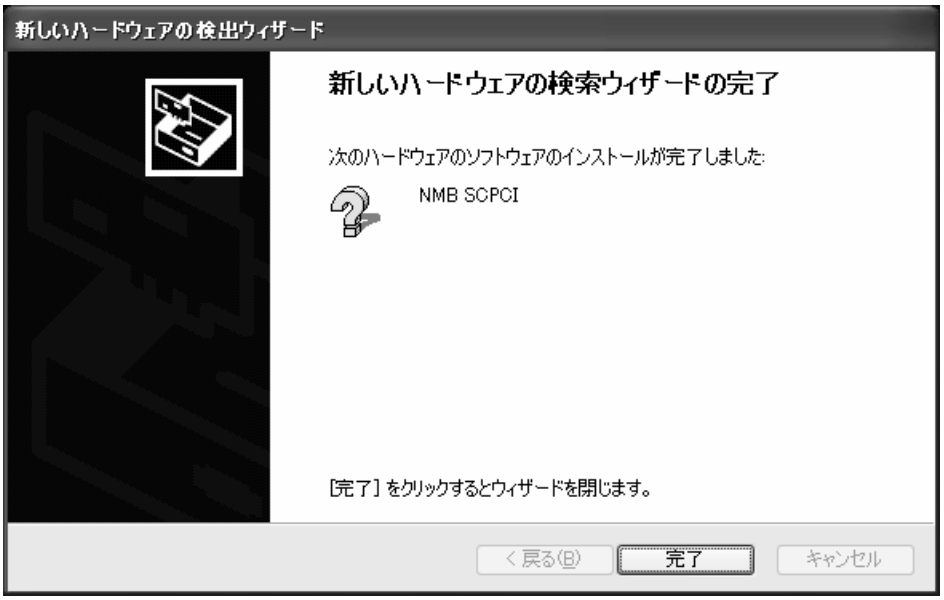


(3)　「次の場所で最適のドライバを検索する」を選択してください。
　　次に、「リムーバブルメディア(フロッピー、CD-ROM など)を検索」を選択してください。

(4)　同梱されている、CD-ROM を、CD-ROM ドライブに挿入してから「次へ(N)」ボタンを
　　クリックして下さい。

(5)　次の画面が表示され、ドライバのインストールが開始されます。

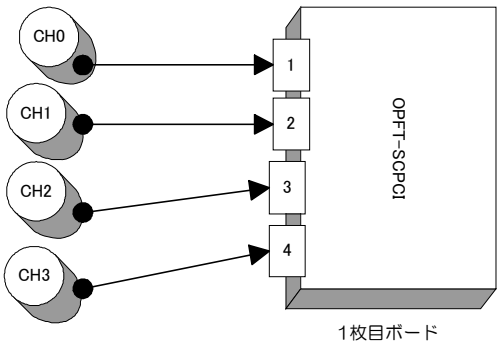
(6)　ドライバのインストールが終了すると、次の画面が表示されます。



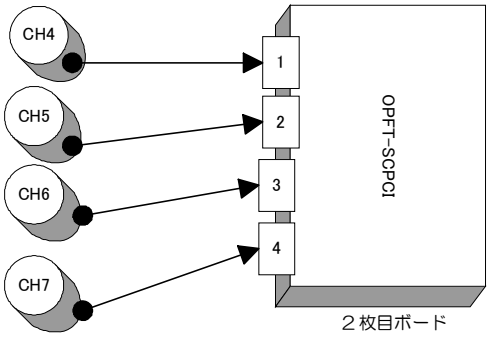
（7）「完了」ボタンをクリックすると、ハードウェアのインストールが終了します。
※2 枚の OPFT-SCPCI ボードが実装されていても、本操作 1 回でインストールは完了します。
注：インストール中に「デジタル署名がありません」というダイアログが表示される事がありますがそのままインストールを続行して下さい。
(8)　CD-ROM ドライブから、CD-ROM を取出して下さい。

2-4. 接続

センサから出ているケーブルを、空いている CH 番号のコネクタに接続します。
これにより、センサに電力が供給されます。



６軸センサ



６軸センサ

3．機能の説明(Windows 対応ドライバ)

3-1． 付属ドライバ CD-ROM 内容

Win フォルダの中に、Disk 及び VBVC フォルダがあります。

（1）Disk フォルダ

scpci.sys	ドライバ
scpci.inf	INF ファイル
UnInstall.exe	ドライバ削除ユーティリティ

（2） VBVC フォルダ

SCPCI.bas	Visual Basic アプリケーション開発用、標準モジュール追加ファイル
scpci.dll	Visual Basic アプリケーション開発用
scpci.h	Visual C++アプリケーション開発用、ヘッダーファイル
scpci.lib	Visual C++アプリケーション開発用

※本ドライバでは、Microsoft Visual C++と Microsoft Visual Basic 用のアプリケーションインタフェース(API)を用意しています。ご使用されるアプリケーション開発言語に応じて使用するアプリケーションインターフェースを選択下さい。

3-2． アプリケーション開発用ファイル使用方法

（1）Visual C++

① scpci.h をヘッダーファイルとして、ソースファイルに追記します。
② プロジェクトのフォルダに、scpci.h、scpci.lib、scpci.dll をコピーします。
③ 「プロジェクト」メニューの「設定」を選択します。
④ 「リンク」を選択し、scpci.lib を設定します。
⑤ 作成した exe ファイルと同じフォルダ内に scpci.dll をコピーし、exe を起動します。

（2）Visual Basic

① プロジェクトのフォルダに、SCPCI.bas、scpci.dll をコピーします。
② SCPCI.bas を、標準モジュールに追加します。
③ 作成した exe ファイルと同じフォルダ内に scpci.dll をコピーし、exe を起動します。

3-3.関数説明

本ボード付属ソフトとして、以下の 4 種類の関数(API)があります。

・ドライバオープン用関数
・ドライバクローズ用関数
・データ送信用関数
・データ受信用関数

本ドライバ使用時は、必ず

① ドライバオープン関数
② データ送信用関数、データ受信用関数
③ ドライバクローズ関数

の順番でご使用下さい。

3-3-1.ドライバオープン用関数

ドライバのオープンを行い、ハンドルを取得し、アプリケーションから本ボードの制御が行える様にします。

① 関数名：DriverOpen()
② 引数　：無し
③ 戻り値：int
　0　→ ハンドル取得成功(正常)
　1　→ ハンドル取得失敗、あるいはハンドル取得済み
　　（二重にオープン操作をしていないか、確認して下さい）
　2　→ 同一のボード番号が存在する
　　（JP5 設定が同一のボードが無いか確認して下さい）
④ サンプルプログラム

（1）Visual C++

```
Int          ret;

ret = DriverOpen();
if( ret == 0 ) {}              // 成功
else if( ret == 1 ) {}         // 失敗
else {}                        // チャンネル異常
```

（2）Visual Basic

```
Dim ret As Integer

ret = Driveropen()
If ret = 0 then                ‘ 成功
ElseIf ret = 1 then            ‘ 失敗
ElseIf ret = 2 then            ‘ チャンネル異常
EndIf
```

3-3-2.ドライバクローズ用関数

ドライバのクローズを行い、ハンドルを開放します。

① 関数名：DriverClose()
② 引数　：無し
③ 戻り値：無し
④ サンプルプログラム

（1）Visual C++

```
DriverClose();
```

（2）Visual Basic

```
DriverClose()
```

3-3-3.データ送信関数

データの送信を行います。
センサに対して、各種コマンド及びデータを送信する為に使用します。

① 関数名：SndPkt(char ch,unsigned long *buf,unsigned int len)
② 引数　：有り
　ch　→　チャネル番号(0 ～ 7)送信するチャネルを指定します。
　buf　→　送信データを格納したバッファ(配列名称の先頭ポインタ)を指定します。
　　バッファサイズは、Long サイズ×64(256 バイト)を確保して下さい。
　len　→　送信データ数(バイト単位：1 ～ 256)を指定します。
　　但し、送信コマンド長は 15 ですのでここでは 15 として下さい。
③ 戻り値：int
　0　→　送信動作完了
　1　→　ボード内ファームウェア送信バッファにデータが残っている為送信失敗
　3　→　異常動作
　　・引数が範囲外
　　・ドライバがオープンされていない
　　・コマンド(番号)が規定外である。
④ 送信データ設定
　送信データは、バッファの先頭に'@'、次にコマンド、データ、最後に':'をセットして下さい。
　・必要に応じたコマンドデータを、バッファに設定して下さい。コマンドデータの詳細は、センサ取扱説明書を参照下さい。

31 24	23 16	15 8	7 0
送信 データ2	送信 データ1	コマンド	@
送信 データ6	送信 データ5	送信 データ4	送信 データ3
送信 データ10	送信 データ9	送信 データ8	送信 データ7
	：	送信 データ12	送信 データ11

※送信データをバッファにセットし、SndPkt()を実行すると指定チャネルに接続されているセンサに対して、バッファ内のデータを送信します。

⑤ サンプルプログラム

（1）Visual C++

```
Int                 ret;
unsigned long       buf[64];

ret = SndPkt(0, buf, 15);
f( ret == 0 ){}                         // 成功
else if( ret == 1 ) {}                  // 失敗
else {}                                 // 異常動作
```

（2）Visual Basic

```
Dim ret As Integer
Dim buf(0 To 63) As Long

ret = SndPkt(0, buf(0), 15)
If ret = 0 then                         ‘ 成功
ElseIf ret = 1 then                     ‘ 失敗
ElseIf ret = 3 then                     ‘ 異常動作
EndIf
```

3-3-4. データ受信関数

データの受信確認あるいは受信データの取得を行います。

① 関数名：GetPkt(char ch,unsigned long *buf,char mode)

② 引数　：有り

ch　　→　チャネル番号(0 ～ 7)受信するチャネルを指定します。

buf　　→　受信データ格納用バッファ先頭ポインタ(サイズは 128Byte を確保して下さい)

mode　→　受信モードの選択(0、0 以外)を指定します。

0　　=　最新データモード

0 以外=　完全データモード

③ 戻り値：int

0　→ 受信データ有り

1　→ 受信データ無し

2　→ 受信エラー（受信ステータスが、0 以外の時セット）

3　→ 異常動作

・引数が範囲外

・ドライバがオープンされていない

④ モード説明

（1）　最新データモード

GetPkt()関数を実行した時の最新データを受信データ格納用バッファにセットします。

（2）　完全データモード

GetPkt()関数を実行する毎に受信したデータを古い順に受信データ格納用バッファにセットします。

⑤ ドライバが確保しているバッファサイズについて

ドライバは、1 枚あたり 0x80000(524288Byte)サイズのバッファをパソコンのメインメモリに確保します。

センサからの 1 回の受信に対して、128Byte を使用する為、524288÷128＝4096 個の受信データを保持出来ます。

OPFT-SCPCI ボードに 4 個のセンサが接続されている場合は、4096÷4＝1024 個でセンサ 1 個あたり 1024 個のデータを保持出来ます。

センサのストリーム出力間隔は、100usec ですので 1024×100usec＝0.1024 秒間のデータを、GetPkt()を実行しなくても保護します。この時間 0.1024 秒が保護時間となります。この時間内に、完全データモードで GetPkt()を実行すれば、バッファ内のデータを欠落無く読み出す事が出来ます。

読み出している間も、センサからのデータを、読み出し終わった所に格納していきます。

但し、保護時間内に GetPkt()を実行しない場合は、その間のバッファ内データは無効となり(バッファは空)、以後の最新データが再度バッファを埋めていきます。

（保護時間内に GetPkt が実行されない場合に、バッファオーバーラン状態となります）

⑥ 受信データ（受信データ格納用バッファイメージ）

31 24	23 16	15 8	7 0
チャネルNo	受信カウンタ	受信 ステータス	受信レングス
受信 データ4	受信 データ3	受信 データ2	受信 データ1
受信 データ8	受信 データ7	受信 データ6	受信 データ5
受信 データ12	受信 データ11	受信 データ10	受信 データ9
受信 データ124	・・・・・・・・・・	受信 データ14	受信 データ13

受信データ格納用バッファには、受信レングス、受信ステータス、…、受信データ 124 個（最大）の順番で 128Byte 分がセットされます。

（1） 受信レングス

'@'から':'までの受信バイト数がセットされます。

（受信データ 1 から受信データ 15 までが通常の受信データとなります）

（2） 受信ステータス[対応 bit がセットされる条件]

センサとの通信状態をボードが判定し、その状態をセットします。

bit0：ストリームモード時、規定時間内にデータが受信されない場合セット

bit1：コマンドエラー（規定のコマンドでない、ターミネータ位置が違う）

bit2：バッファオーバーラン時セット

bit3：フレーミングエラー時セット

※受信ステータスがヌル(00H)時は、本ボードが検出できるエラーが発生していない場合です。

※bit0 がセットされている場合は、センサケーブル断線等が考えられます。

アプリケーションを停止させて、センサケーブルを確認して下さい。

（3） 受信カウンタ

センサからデータを受信した回数がセットされます。範囲は、0～255 で 255 の次は、0 となります。

（4）チャネル No

ボードがデータを取込んだチャネルの番号がセットされます。範囲は、0～7 です。
このチャネル No が、引数で指定したチャネル番号と一致しない場合は、データが異常となりますので、再度のデータ受信を行う様にして下さい。

⑦ サンプルプログラム

（1）Visual C++

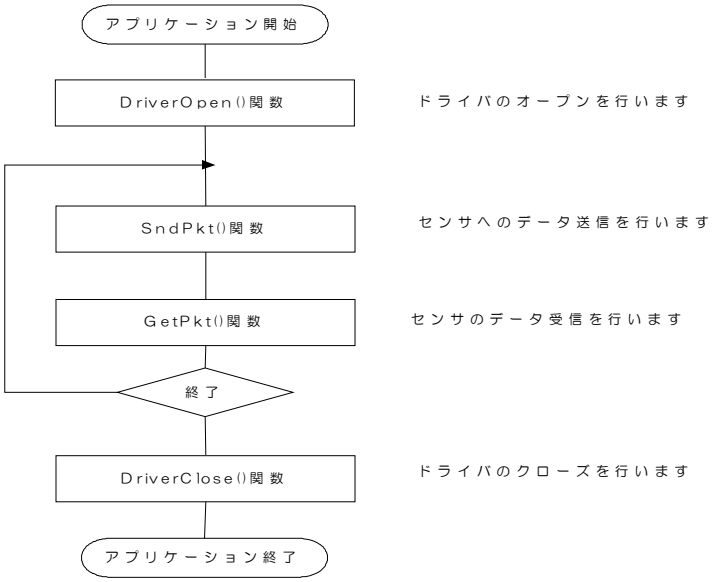
```
unsigned long        ret, buf[32];

ret = GetPkt(0, buf);
if( ret == 0 ) {}                    // 受信データあり
else if( ret == 1 ) {}               // 受信データなし
else if( ret == 2 ) {}               // エラーあり(buf のデータの“受信ステータス”を確認)
else {}                              // パラメータ異常
```

（2）Visual Basic

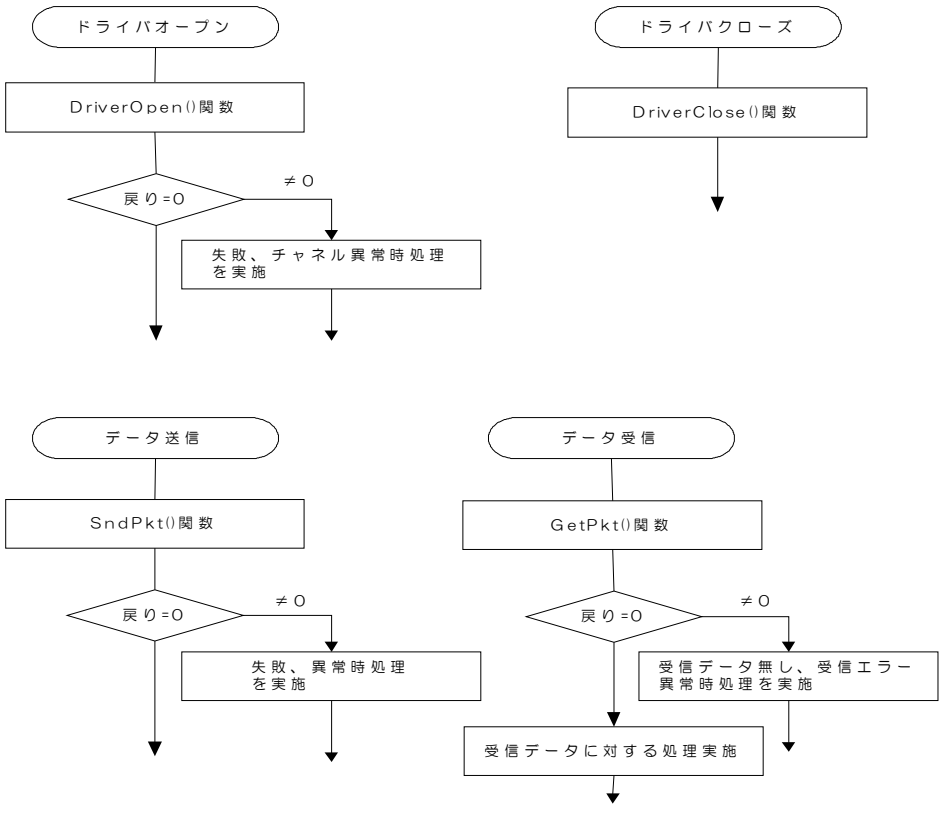
```
Dim ret As Long
Dim buf(0 To 31) As Long

ret = GetPkt(0, buf(0))
If ret = 0 then                      ‘ 受信データあり
ElseIf ret = 1 then                  ‘ 受信データなし
ElseIf ret = 2 then                  ‘ エラーあり(buf のデータの“受信ステータス”を確認)
ElseIf ret = 3 then                  ‘ パラメータ異常
EndIf
```

3-4. 各関数の使用シーケンス例

(1) 基本シーケンス



(2) 各関数



４．機能の説明(Linux 対応ドライバ)

４-１． 付属ドライバ CD-ROM 内容

Linux フォルダの中に、ForceSensor フォルダがあります。

(1)ForceSensor フォルダ

Makefile	カーネル 2.4.x用 Makeファイル
scpci.h	ヘッダーファイル
plx9054.h	ヘッダーファイル
scpci.c	ドライバ・ソースファイル
app1.c	サンプルアプリケーション1・ソースファイル
app2.c	サンプルアプリケーション2・ソースファイル
app3.c	サンプルアプリケーション3・ソースファイル
app4.c	サンプルアプリケーション4・ソースファイル
Makefile.linux26	カーネル 2.6.x用 Makeファイル
Makefile.rt	RT Linux用 Makeファイル
scpci_rt.c	RT Linux用 サンプルアプリケーション・ソースファイル
monitor_rt.c	RT Linux用 サンプルアプリケーション・ソースファイル

※ このサンプル・ドライバでは、カーネル 2.4.x 用、カーネル 2.6.xMakeFile 及び RT Linux 用サンプルを分けて用意しています。

４-２． 関数説明

本ボード付属ソフトとして、以下の関数(API)を用意しています。これらの API はすべて scpci.h の後半部分に定義されていますので、併せてご参照ください。

４-２-１． ドライバオープン用関数

/dev/scpci デバイスファイルをオープンし，ドライバに初期化命令を発行します。

①関数名：OpenDriver()
②引数　：なし
③戻り値：int
　　デバイスファイルデスクリプタ
④サンプル

```
fd = OpenDriver();
if( fd < 0){
    printf("Driver Open Error. Exiting..¥n");
    exit(0);
}
```

4-2-2. ドライバクローズ用関数

/dev/scpci デバイスファイルをクローズします。

①関数名：CloseDriver(int fd)

②引数　：

fd → デバイスファイルデスクリプタ[OpenDriver 関数の戻り値(fd)]

③戻り値：なし

④サンプル

CloseDriver(fd);

4-2-3. 全センサの通信開始用関数

全チャネルのセンサに対して，送信開始コマンドを送信し、受信を開始します。
ドライバを組み込んだ時点で本関数は実行されるので、必要な場合に使用します。
(通信停止用関数を実行した後に通信を再開する等)

①関数名：StartForceCapture(int fd)

②引数　：

fd → デバイスファイルデスクリプタ[OpenDriver 関数の戻り値(fd)]

③戻り値：なし

④サンプル

StartForceCapture(fd);

4-2-4. 全センサの通信停止用関数

全チャネルのセンサに対して，送信停止コマンドを送信し、受信を停止します。

①関数名：StopForceCapture(int fd)

②引数　：

fd → デバイスファイルデスクリプタ[OpenDriver 関数の戻り値(fd)]

③戻り値：なし

④サンプル

StartForceCapture(fd);

4-2-5. 各チャネル(センサ)のレンジ取得用関数

指定したチャネルのセンサのレンジデータを取得するコマンドを送信し、指定したセンサのレンジデータを受信し、レンジデータを取込みます。

①関数名：GetRangeData(int fd, int channel, int Frange[6])

②引数　：

fd　→ デバイスファイルデスクリプタ[OpenDriver 関数の戻り値(fd)]

channel　→ チャネル番号(0 ～ 7)でレンジデータを読込むチャネルを指定します。

Frange[6] → 読込んだレンジ値をセットします。

Frange[0] ← Fx、Frange[1] ← Fy、Frange[2] ← Fz

Frange[3] ← Mx、Frange[4] ← My、Frange[5] ← Mz

※ 但し、レンジ値は Fx = Fz=Fy、Mx=My=Mz、これらの値はセンサ固有の値です。従って必ず取り込む必要があります。

③戻り値：int

0　→ 成功

-1　→ 失敗(タイムアウト)　マクロ変数 SENSOR_TIMEOUT

-2　→ 失敗(センサ未接続)　マクロ関数 SENSOR_NONE

④サンプル

```
ret=GetRangeData( fd, ch, force_range[ch]);
```

4-2-6. 各チャネル(センサ)の移動平均回数読込み用関数

指定したチャネルのセンサの移動平均回数を取得するコマンドを送信し、指定したセンサの移動平均回数を取込みます。

①関数名：GetMovingAverage(int fd, int channel)

②引数　：

fd　→ デバイスファイルデスクリプタ[OpenDriver 関数の戻り値(fd)]

channel　→ チャネル番号(0 ～ 7)で移動平均回数を読込むチャネルを指定します。

③戻り値：int

1 ～ 100　→ 成功(移動平均回数の現在値)

-1　→ 失敗(タイムアウト)　マクロ変数 SENSOR_TIMEOUT

-2　→ 失敗(センサ未接続)　マクロ関数 SENSOR_NONE

④サンプル

```
ret= GetMovingAverage ( fd, ch);
```

4-2-7. 各チャネル(センサ)の移動平均回数設定用関数

指定したチャネルのセンサに移動平均回数を登録するコマンドを送信し、指定したセンサの移動平均回数を書換えます。

①関数名：SetMovingAverage(int fd, int channel, int N_moving_average)

②引数　：

fd　→ デバイスファイルデスクリプタ[OpenDriver 関数の戻り値(fd)]

channel　→ チャネル番号(0 ～ 7)で移動平均回数を書込むチャネルを指定します。

N_moving_average　→ 移動平均回数の設定値(1 ～ 100)

③戻り値：int

0　→ 常に 0 が戻ります。

④サンプル

ret = SetMovingAverage(fd, ch, ave);

4-2-8. 各チャネル(センサ)の A/Z 機能を有効設定用関数

指定したチャネルのセンサの A/Z 機能を実行(A/Z ON)するコマンドを送信します。
この関数を実行した時点の力を原点とします。

①関数名：SetAutoZeroFunc(int fd, int channel)

②引数　：

fd　→ デバイスファイルデスクリプタ[OpenDriver 関数の戻り値(fd)]

channel　→ チャネル番号(0 ～ 7)で A/Z を実行するチャネルを指定します。

③戻り値：int

0　→ 常に 0 が戻ります。

④サンプル

ret= GetMovingAverage (fd, ch);

4-2-9. 各チャネル(センサ)の A/Z 機能の無効設定用関数

指定したチャネルのセンサの A/Z 機能を無効(A/Z OFF)にするコマンドを送信します。

①関数名：UnsetAutoZeroFunc(int fd, int channel)

②引数　：

fd　→ デバイスファイルデスクリプタ[OpenDriver 関数の戻り値(fd)]

channel　→ チャネル番号(0 ～ 7)で A/Z を無効にするチャネルを指定します。

③戻り値：int

0　→ 常に 0 が戻ります。

④サンプル

ret= UnsetAutoZeroFunc (fd, ch);

4-2-10. 各チャネル(センサ)の最新データ読込み用関数

指定したチャネルの最新データを取込みます。

①関数名：GetForce(int fd,　int channel, int Force[6])

②引数　：

fd　→ デバイスファイルデスクリプタ[OpenDriver 関数の戻り値(fd)]

ch　→ チャネル番号(0 ～ 7)でデータ取込みを行うチャネルを指定します。

Force → 最新の力データをセットするメモリを指定します。

Force[0] = Fx、Force[1] = Fy、Force[2] = Fz

Force[3] = Mx、Force[4] = My、Force[5] = Mz

③戻り値：int

0　→ 常に 0 が戻ります。

-2 → 失敗(センサ未接続)　マクロ関数 SENSOR_NONE

④サンプル

ret= GetForce (fd, ch、force);

4-2-11．全チャネル(センサ)のデータ読込み用関数

全チャネルの最新のカデータの取込を行う為の関数です。

①関数名：GetAllChannelForce(int fd,　struct force_data *force)

②引数　：

fd　→ デバイスファイルデスクリプタ[OpenDriver 関数の戻り値(fd)]

force → 最新のカデータをセットする構造体を指定します。

③戻り値：なし

force_data 構造体の status メンバ変数の値が

NO_SENSOR_STATUS(=0xFFFFFFFF)の時、該当チャネルセンサ未接続。

④サンプル

```
GetAllChannelForce ( fd, force);
```

⑤構造体

```
//各チャネルのセンサ情報格納用構造体
struct force_data{
        int channel;
        int F[6];           // = {Fx、Fy、Fz、Mx、My、Mz}
        int Frange[6];      //for {Fx、Fy、Fz、Mx、My、Mz}
        int N_moving_average;
        unsigned int status;
        unsigned char seqnum;
        unsigned char previous_seqnum;
};
```

4-2-12. 各チャネル(センサ)の完全データ読込み用関数

指定チャネルのカデータを取りこぼし無く取り込む為の関数です。

①関数名：GetForceComplete(int fd,　int channel, int Force[6])

②引数　：

fd　→ デバイスファイルデスクリプタ[OpenDriver 関数の戻り値(fd)]

ch　→ チャネル番号(0 ～ 7)でデータ取込みを行うチャネルを指定します。

Force → 最新のカデータをセットするメモリを指定します。

③戻り値：int

(0 ～ 255) → 成功(0 から 255 の連続番号、0 の次は 1、255 の次は 0)

-1　→ 読出し待ちデータなし。 マクロ変数 SENSOR_NO_DATA

-2　→ センサ未接続。　　マクロ変数 SENSOR_NON

④サンプル

ret = GetForceComplete(fd,ch, force[ch]);

⑤その他

この関数は、リングバッファ上に格納されている受信済みのデータを、順次読出しデータの取りこぼしを防ぎます。

センサのデータは、100usec 毎にセンサより送信され、DMA によりメインメモリ上のリングバッファに格納されています。

このため、この関数でデータを読む為には、センサのデータ送信間隔よりも平均的に早いタイミングで読み出す必要があります。

4-2-13. 各チャネル(センサ)のデータ読込み用関数

指定チャネルのカデータを配列(128Byte)に返す関数です。

①関数名：GetPkt(int fd, int channel, char *buf,int mode)

②引数　：

fd　→ デバイスファイルデスクリプタ[OpenDriver 関数の戻り値(fd)]

ch　→ チャネル番号(0 ～ 7)でデータ取込みを行うチャネルを指定します。

*buf　→ カデータをセットする配列を指定します。

配列の(0)～(14)にセンサから受信したデータがセットされます。

mode → モードを指定します。

0：最新データモード(最新のデータを返します)

0 以外：完全データモード(取りこぼしの無いデータを返します)

③戻り値：int

0　→　成功

-1　→ 読出し待ちデータなし。

(完全データモード) マクロ変数 SENSOR_NO_DATA

-2　→ センサ未接続。　　マクロ変数 SENSOR_NON

④サンプル

ret = GetPkt(fd, ch, buf, 1);

4-2-14. 各チャネル(センサ)に対する配列データの送信用関数

指定チャネルのセンサに、配列(128Byte)のデータを送信する関数です。
任意のコマンドをセンサに送るのに使用します。

①関数名：SndPkt(int fd, int channel, char *buf,int length)

②引数　：

fd　→ デバイスファイルデスクリプタ[OpenDriver 関数の戻り値(fd)]

ch　→ チャネル番号(0 ～ 7)でデータ書込みを行うチャネルを指定します。

*buf　→ (書込み)データをセットする配列を指定します。

lenght→ 15 固定(書込みデータ数を指定)

③戻り値：int

0　→　常時　0 を返します

④サンプル

```
PACKET_MEASURE_MODE="@A¥0¥0¥0¥0¥0¥0¥0¥0¥0¥0¥0:"
ret = SndPkt(fd, 0, PACKET_MEASURE_MODE, 15 );
```

4-3. RT Linux API 一覧

プロトタイプ宣言は、scpci.hで行っているので 必ずincludeする必要があります。

GetRangeData()
GetForce ()
SetAutoZeroFunc ()
UnsetAutoZeroFunc ()
GetMovingAverage ()
SetMovingAverage ()

書式：
```
int GetRangeData(int ch, int Frange[6]);
int GetForce(int ch, int Force[6]);
int SetAutoZeroForce(int ch);
int UnsetAutoZeroForce(int ch);
int GetMovingAverage(int ch);
int SetMovingAverage(int ch, int N_moving_average);
```

内容、戻値：　前出のAPIと動作は同じ(サンプル　scpci_rt.c も参照ください)です。

備考：

リアルタイムプロセスはカーネルモジュールとして作成されます。このAPIはリアルタイムプロセスのプログラム中で呼び出すことができますが、あらかじめデバイスドライバscpci.oを組み込んでおくことが必要となります。

4-4. ioctl 一覧

デバイスドライバとユーザプロセス間のやり取りは全てioctlにより行ないます。
前の章で紹介したAPIの実際の処理は，scpci.h後半に定義されているとおり，ここに列挙するioctlによりドライバとやりとりしています。

4-4-1. IOCTL_SCPCI_OPEN

①書式　：ioctl(fd, IOCTL_SCPCI_OPEN);
②内容　：ドライバをオープンした際のドライバへの初期化発行です。これにより完全データモードによる読込みのための読込ポインタ関連の変数がリセットされます。又、新たに追加接続されたセンサに対応するため、送信開始コマンドを全チャネルのセンサに送信します。

4-4-2. IOCTL_GET_FORCE_CHANNEL

①書式　：ioctl(fd, IOCTL_GET_FORCE_CHANNEL, &force);
②内容　：forceはforce_data構造体(scpci.hに定義)です。force_data構造体のメンバ変数channelに取得したいチャネル番号を代入後に、このioctl命令を発行します。
ドライバは当該チャネルの各軸力データ、及び各軸レンジ値、フィルタ設定値メンバ変数（メンバ変数の定義は前章APIのGetAllChannelForce関数を参照）に値を代入して戻します。
戻される力データは受信済データの最新値（最新データモード）です。

4-4-3. IOCTL_GET_FORCE_ALL

①書式　：ioctl(fd, IOCTL_GET_FORCE_ALL, force);
②内容　：force=struct force_data force[8]へのポインタの指定で行います。
全チャネルの最新力データをドライバから取得（最新データモード）。
基本的な使い方は上(IOCTL_GET_FORCE_CHANNEL)と同じです。

4-4-4. IOCTL_GET_FORCE_COMPLETE

①書式　：ioctl(fd, IOCTL_GET_FORCE_COMPLETE, &force);
②内容　：forceはforce_data構造体(scpci.hに定義)です。
力データを取りこぼしなく取得する（完全データモード）。
基本的な使い方は上(IOCTL_GET_FORCE_CHANNEL)と同じです。

4-4-5. IOCTL_START_CAPTURE

①書式　：ioctl(fd, IOCTL_START_CAPTURE);
②内容　：開始コマンドの送信
全チャネルのセンサに対して開始コマンドを送信し，インタフェースボードのデータキャプチャも開始します。

4-4-6. IOCTL_STOP_CAPTURE

①書式　：ioctl(fd, IOCTL_STOP_CAPTURE);
②内容　：停止コマンドの送信
全チャネルのセンサに対して停止コマンドを送信し，インタフェースボードのデータキャプチャも停止します。

4-4-7. IOCTL_AUTOZERO_ON

①書式　：ioctl(fd, IOCTL_AUTOZERO_ON, &channel);

②内容　：A/Z ON動作

channelに対象とするch番号を代入してから発行します。
当該チャネルのセンサに対してオートゼロ有効に設定するためのコマンドをセンサに送信します。

4-4-8. IOCTL_AUTOZERO_OFF

①書式　：ioctl(fd, IOCTL_AUTOZERO_OFF, &channel);

②内容　：A/Z OFF動作

channelに対象とするch番号を代入してから発行します。
当該チャネルのセンサに対してオートゼロ無効に設定するためのコマンドをセンサに送信します。

4-4-9. IOCTL_GET_SENSOR_SPEC

①書式　：ioctl(fd, IOCTL_GET_SENSOR_SPEC, &channel);

②内容　：channelに対象とするch番号を代入してから発行します。
ドライバは当該チャネルのセンサに対してレンジデータとフィルタ設定値を要求するコマンドを送信し、返答を待ちます。
その後にIOCTL_GET_FORCE_CHANNEL命令によってforce_data構造体を取得することでレンジ値やフィルタ値を得ることができます。

4-4-10. IOCTL_SET_MOVINGAVE

①書式　：ioctl(fd, IOCTL_SET_MOVINGAVE, &force);

②内容　：force_data構造体forceのchannel番号とフィルタ値N_moving_averageに変更したい値を代入後にこのioctlを発行すると、ドライバは当該センサに変更用のコマンドを送信します。

4-4-11. IOCTL_GET_PKT_COMPLETE

①書式　：IOCTL_GET_PKT_COMPLETE

②内容　：完全データモード

4-4-12. IOCTL_GET_PKT_CURRENT

①書式　：IOCTL_GET_PKT_CURRENT

②内容　：最新データモード

4-4-13. IOCTL_SND_PKT

①書式　：ioctl(fd, IOCTL_SND_PKT, &packet);

②内容　：配列単位でのデータのやり取りのために使用します。

4-5. アプリケーション　サンプルプログラム

4-5-1. サンプル1

※app1.c (% gcc –o app1 app1.c)

```
#include <stdio.h>
#include <stdlib.h>
#include "scpci.h"
main() {
    int fd;
    struct force_data force[8];

    fd = OpenDriver();
    if( fd < 0){
        printf("Driver Open Error. Exiting..¥n");
        exit(0);
    }
    while(1){
        //Get All Newest Channel Force into structure
        GetAllChannelForce( fd, force );
        //Print out for ch0 ..
        if( force[0].status == NO_SENSOR_STATUS ) printf("[1] No Sensor, ");
        else
            printf("[1]Fx=%6.1f Fy=%6.1f Fz=%6.1f (%3d:%8x), ",
                ((double)force[0].F[0]/32768.0) *(double)force[0].Frange[0],
                ((double)force[0].F[1]/32768.0) *(double)force[0].Frange[1],
                ((double)force[0].F[2]/32768.0) *(double)force[0].Frange[2],
                force[0].seqnum, force[0].status
            );
    }
    CloseDriver(fd);
}
```

内容：

全チャネル(8ch)の最新の力データを構造体配列force[8] に格納。その構造体のチャネル1のセンサ内容について実際の力への換算値などをプリント出力します。

4-5-2. サンプル2

※app2.c (% gcc –o app2 app2.c)

```
#include <stdio.h>
#include <stdlib.h>
#include "scpci.h"
main() {
    int fd;
    int i,ret;
    int force[6], force_range[MAX_CHANNEL][6], N_moving_average;

    fd = OpenDriver();
    if( fd < 0){
        printf("Driver Open Error. Exiting..¥n");
        exit(0);
    }

    for(i=0;i<MAX_CHANNEL;i++){
        ret=GetRangeData( fd, i, force_range[i]);
        N_moving_average=GetMovingAverage(fd, i);
        if(ret==SENSOR_NONE) printf("[Ch%d] No Sensor connected..¥n",i);
        else if(ret==SENSOR_TIMEOUT) printf("[Ch%d] Timeout..¥n",i);
        else{
                printf("[Ch%d] Frange[0],[1],[2]=%d¥t, Frange[3],[4],[5]=%d¥t, N=%d¥n",i,
                        force_range[i][0],// Fx,Fy,Fz range are same value
                        force_range[i][3],// Mx,My,Mz range are same value
                        N_moving_average
                );
        }
    }
    //Get Each Channel Force into array force[6]
    ret=GetForce( fd, 0, force );
    printf("[1]Fx=%6.1f Fy=%6.1f Fz=%6.1f (%3d), ",
        ((double)force[0]/32768.0) *(double)force_range[0][0],
        ((double)force[1]/32768.0) *(double)force_range[0][1],
        ((double)force[2]/32768.0) *(double)force_range[0][2],
        ret
    );
}
```

内容：

最初のfor文で全チャネルのレンジデータ，フィルタ設定を取得後，プリントを行う。
最後に，チャネル0の最新力データを配列force に取得，プリント出力して終了。

4-5-3. サンプル3

※app3.c (% gcc –o app3 app3.c)

```
#include <stdio.h>
#include <stdlib.h>
#include "scpci.h"
main() {
    int fd;
    int ret,i,j;
    struct force_data force;
    char buf[128];//for one packet
    Rx_Header_Register *header;
    DataPacketFormat *recv_data;

    fd = OpenDriver();
    if( fd < 0){
         printf("Driver Open Error. Exiting..¥n");
         exit(0);
    }
    for( j=0;j<100000;j++){
         // for channel 1
         while(1){
                   ret=GetPkt( fd, 0, buf, 1 );
                   if(ret==0||ret==SENSOR_NONE) break;
         }
         if( ret == 0){
                   header=(Rx_Header_Register *)&buf;
                   recv_data=(DataPacketFormat *)(header+1);
                   printf("[0]Header(4Bytes)=0x%8x (ch=%d, Seq.No.:%3d ) ",
                             header->All, header->Bits.chnum, header->Bits.seqnum );
                   }
                   if( ret == SENSOR_NONE){
                             printf("[0] No sensor connected.. ");
                   }
         }
}
```

内容：

チャネル0のセンサから受信するデータを配列単位で取得します。
上記プログラムでは完全データモードでバケットを取得しています。
（Rx_Header_Register，DataPacketFormat構造体の定義は，scpci.hを参照．）

4-5-4. サンプル4

※app4.c (% gcc –o app4 app4.c)

```
#include <stdio.h>
#include <stdlib.h>
#include "scpci.h"
main() {
   int fd;
   int chk,i,j,all_channel_read;
   int ret[MAX_CHANNEL],seqnum[MAX_CHANNEL],seqnumOld[MAX_CHANNEL];
   int lost_counter[MAX_CHANNEL],lost_total=0;
   int force[MAX_CHANNEL][6];

   fd = OpenDriver();
   if( fd < 0){
        printf("Driver Open Error. Exiting..¥n");
        exit(0);
   }
   for( j=0;j<10000000;j++){
        for( i=0;i<MAX_CHANNEL;i++){
                while(1){
                        seqnum[i]=GetForceComplete( fd, i, force[i] );
                        if( seqnum[i]>=0) break;
                        else if( seqnum[i] == SENSOR_NONE ) break;//No Sensor.
                }
        }
        //Sequence Number Check:
        printf("%5d [1] %3d [2] %3d [3] %3d [4] %3d [5] %3d [6] %3d [7] %3d [8] %3d",
                j,seqnum[0],seqnum[1],seqnum[2],seqnum[3],seqnum[4],seqnum[5],seqnum[6],
                seqnum[7]);
        //Force Data Check:
        chk=0;//select axis.. 0-5=Fx Fy Fz Mx My Mz
        //printf("%5d [1] %3d [2] %3d [3] %3d [4] %3d [5] %3d [6] %3d [7] %3d [8] %3d",
        // j,force[0][chk],force[1][chk],force[2][chk],force[3][chk],
        // force[4][chk],force[5][chk],force[6][chk],force[7][chk]);
        //Each Channle Data:
        chk=0;//select channel
        //printf("%5d Fx=%5d Fy=%5d Fz=%5d Mx=%5d My=%5d Mz=%5d",
        //j,force[chk][0],force[chk][1],force[chk][2],force[chk][3],force[chk][4],force[chk][5] );
   }
}
```

内容：

全チャネルのカデータを取りこぼしなく取得。全チャネルのシーケンス番号のみ出力。
(コメント行に、各チャネル・各軸のカデータの出力例も記述)

4-5-5. サンプル5

※RTLinux のリアルタイムモジュールサンプル scpci_rt.c

```
#include <rtl.h>
#include <rtl_fifo.h>
#include <rtl_sched.h>
#include <rtl_sync.h>
#include <time.h>
#include <pthread.h>
pthread_t thread;
#include <math.h>
#include "scpci.h"
#define SAMPLING_TIME 0.001//[sec]
int fifo_size=4000;
int force[6], force_range[6];

void * start_routine(void *arg)
{
   struct sched_param p;
   double st=SAMPLING_TIME;
   int header;
   p . sched_priority = 1;
   pthread_setschedparam (pthread_self(), SCHED_FIFO, &p);
   pthread_make_periodic_np (pthread_self(), gethrtime(), (int)(st*1000000000));
   while (1) {
         pthread_wait_np ();
         //Newest Force Data is obtained.
         header=GetForce(0, force);
         rtf_put( 0 , &force, sizeof(force));
   }
   return 0;
}
```

```
int init_module(void)
{
   int n;
   pthread_create(&thread, NULL, start_routine, 0);
   pthread_setfp_np(thread, 1);
   rtf_create(0, fifo_size);
   //Range Data is obtained here.
   GetRangeData(0, force_range);
   n=GetMovingAverage( 0 );
   rtl_printf("SCPCI-RTLinux: Range(ch0) Fx,Fy,Fz=%d Mx,My,Mz=%d, N=%d¥n",
                force_range[0],force_range[3], n);
   //A/Z Fuction Enabled for ch0..
   SetAutoZeroFunc(0);
   rtl_printf("SCPCI-RTLinux: Module Installed...¥n");
   return 0;
}


void cleanup_module(void) {
   rtl_printf("SCPCI-RTLinux: Module Removed...¥n");
   rtf_destroy(0);
   pthread_delete_np (thread);
}
```

内容：　（組込み手順は、4-8章を参照）

リアルタイムモジュールを組込む前に、デバイスドライバscpci.oを先に組込んで下さい。
scpci_rt.cは scpciドライバから１ミリ秒毎に最新のカデータを取得しているリアルタイムプロセスのサンプル「前出のapp1.c～app4.c」も同時に使うことができます。
また、ユーザプロセスのプログラムmonitor_rt.cにより/dev/rtf0を通してリアルタイムプロセスが正常にデータを獲得できているかのモニタリングも可能です。
（モニタリングプログラム、monitor_rt.c参照）

4-6. デバイスファイルの作成(リアルタイム Linux では無い場合に必須)

本ドライバでは、ドライバ名を"scpci"とし、同名のデバイスファイルを/dev以下に作成する必要があります。

手動で行うには、以下のコマンド、(添付のMakefileを使用)

```
# make device
```

あるいは生成コマンドから直接作成します。

```
# mknod -m 666 /dev/scpci c 240 0
```

上記、666はデバイスへのアクセスパーミッション（ここでは誰でも読み書きアクセス可に指定)、240番と0番はそれぞれメジャー番号・マイナー番号といい、各デバイスの識別番号のようなIDです。

したがって、その他のデバイスと重複する場合には番号を変更する必要があります。

本ドライバでは240番、マイナー番号は0固定と定めていますが、もし変更する場合にはMakefileの冒頭部分、およびscpci.hのPCI_MAJORマクロ変数でメジャー番号を適当に変更することが可能です。

リアルタイムLinuxでは、リアルタイムプロセスはカーネルモジュール形式で作成されます。したがって、その場合には仮想デバイスファイルを通してハードウェアにアクセスする必要はないので、デバイスファイルの作成は不要となります。

[補足] udevを用いたシステムの場合（例えば、Linux Fedora Core 3 以降のバージョン）

udevのシステムを用いたLinuxでは、/dev/以下のデバイスファイルは起動時に自動的に作成されるような仕様変更が行われています。したがって、上記の方法でデバイスを作成しても、再起動後には無くなってしまいます。
毎回作成しないためには、udevの仕組みに沿って作製する事になります。
最も簡便な方法は、/etc/udev/devicesディレクトリにデバイスファイルを置く事です。
すなわち、

```
# mknod -m 666 /etc/udev/devices/scpci c 240 0
```

を一度だけ実行して/etc/udev/devices以下にscpciデバイスファイルを作成します。
すると、起動時にきちんと毎回/dev以下にコピーする事になります。

4-7. センサ・OPFT-SCPCI 基板のやり取りの流れ

センサとOPFT-SCPCI基板(インターフェースボード)は基本的にRS422の通信でありセンサから受けたデータから力データやセンサ仕様などを取り出すなどのセンサ固有の処理まで含めた処理を本ドライバ側で行っています。
センサからのデータは一定間隔（100μs）で送出されており、ボードで受信されます。
インターフェースボードは獲得データを即時DMA転送によってメインメモリ上のリングバッファに転送しています。
ドライバではFPGAの適切な設定とDMA転送の初期設定の後、DMA転送が完了する毎に発生する割り込み処理の中で、DMA転送されたデータを管理しています。
その管理情報に基づきリングバッファを参照すれば現在の最新データやデータを取りこぼし無く完全に獲得することなどができます。
アプリケーションプログラムから/dev/scpciを通して、データの獲得関連や設定などのioctlをドライバに送ることでセンサとのやりとりが可能となります.。

FPGA：OPFT-SCPCI 基板上の部品を示しています。
DMA ：OPFT-SCPCI 基板上の部品に対して、機能を有効とすることによりメインメモリへのデータ DMA 転送を実現しています。

4-8. コンパイル、ドライバ組込み手順

4-8-1. ドライバの組込み時の内部処理

1．インターフェースボードの検索
2．リングバッファメモリ領域の確保(ボード毎に128Bytes x 4096 個分の領域)
3．DMA転送設定、割り込み開始、A/Z機能有効化
4．全チャネルのセンサへ開始命令送信（コマンド送信）、キャプチャ+DMA開始
5．レンジデータ＋移動平均フィルタ設定値の問い合わせコマンド送信
　その返答は、割り込み処理でキャプチャして取得

4-8-2. アプリケーションによる読込み時の内部処理

1．OpenDriver関数で/dev/scpcをオープンし、初期化命令のioctlをドライバへ通知
2．ドライバは、全チャネルに開始命令を送信し、関連ポインタを初期化する。
　つまり、ドライバモジュール組込み後に、センサの取り外しなどを行っても特に問題とならないが、再接続時の一度だけ送信開始にわずかなdelayが発生します。（再接続したチャネルのみ）
3. GetForce関数、GetForceComplete関数、GetPkt関数などでDMAバッファ上のデータを取得。
　ドライバモジュールをバイナリオブジェクトとして提供することもできます。
　バイナリオブジェクトの場合は、カーネルバージョンに注意する必要があります。
　基本的にLinux-2.4とLinux-2.6系列では互換性はありませんし、2.4/2.6のそれぞれにおいてもバージョンによっては組み込めないかもしれません。
　異なるカーネルバージョンでコンパイルされたモジュールを組み込む際には
　# insmod –f scpci.o ← insmod –f scpci.ko (Linux-2.6系列の場合)
　のように強制的にモジュールを組込む-fオプションを付けてください。
　カーネルバージョンが近ければたいていの場合問題はありません。（カーネル内部の関連する仕様が変更なければ動作します）
　上記の方法をとらない場合は、ソースから[ドライバの組込み手順]で示す手順にてモジュールを作成することになります。
　モジュールのコンパイルには基本的にカーネルヘッダが必要となるので、少々ユーザ側に知識を要求する可能性があります。

4-8-3. Linux Kernel 2.4/2.6 ドライバ組込み手順及びアプリケーション実行

(1)ドライバの組込み手順

＊ Makefile を修正：カーネルヘッダの所在を指定 KERNDIR

（Linux2.6の場合： Makefile.linux26）

＊ モジュールのコンパイル

% make <----Kernel 2.4

or

% make -f Makefile.linux26 <----Kernel 2.6

＊ デバイスファイルの作成：/dev/scpci

% su

make device

＊ モジュールの組込み

insmod scpci.o <----Kernel 2.4 # insmod -f scpci.o （強制組込み）

or

insmod scpci.ko <----Kernel 2.6 # insmod -f scpci.ｋo

(2)アプリケーションの実行

./app1

For more details, see source code...

scpci.c <--Module Source

scpci.h <--Main Header

plx9054.h <--Required for scpci.c

appX.c <--User Application Samples

4-8-4. RT Linux でのドライバ組込み手順及びアプリケーション実行

※RT Linux3.2 [real-time Kernel2.4]

* Edit Makefile.rt for your location of RTLinux Directory
* Real-Time Module Compilation
% make -f Makefile.rt
* RTLinux
% su
# rtlinux start
* Install Sample RealTime Process
# insmod scpci.o <----- or, # insmod -f scpci.o （強制組込み）
# insmod scpci_rt.o
* Check
# dmesg
  or, execute sample monitor program through FIFO /dev/rtf0
./monitor_rt
  or, execute sample program app1, app2, app3, app4
./app4
For more details, see source code...
scpci_rt.c <--RealTime Module
scpci.c <--Driver Object
scpci.h <--Main Header
plx9054.h <--Required only for scpci.c
monitor_rt.c <--User Process for monitoring

備考： RTLinux 向けの API は、先に scpci.o ドライバモジュールを組込んでおかないと使えないので注意が必要です。
ドライバモジュールは通常の Linux のモジュールと共用できます。
完全データモード゛に対応する API は無いので、レンジや最新力データの獲得のための、
int GetRangeData(int ch, int Frange[6]);
int GetForce(int ch, int Force[6]);
int SetAutoZeroFunc(int channel);
int UnsetAutoZeroFunc(int ch);
int GetMovingAverage(int ch);
int SetMovingAverage(int ch, int N_moving_average);
を使用してください。
（使い方は冒頭の RTLinuxAPI の説明、およびサンプルプログラム scpci_rt.c を参照し下さい）

5．ドライバのアンインストール

5-1．ドライバのアンインストール(Windows 版　ドライバ)

(1)　同梱されている、CD-ROM を、CD-ROM ドライブに挿入してさい。

(2)　CD-ROM の Win¥DISK フォルダ内、UnInstall.exe を実行してください。

(3)　次の画面が表示されます。

図 7　アンインストール画面

(4)　「OK」を実行すると、本ボードのドライバは削除されます。
「キャンセル」を実行すると、ドライバの削除は行いません。

5-2．ボードの取り外し(Windows 版、Linux 版　ドライバ共通)

(1)　ボードを取り外す前に、システムをシャットダウンし、コンセントからパソコンの電源ケーブルを抜いてください。

(2)　パソコン本体からケーブル類を外し、カバーを取り外して下さい。

(3)　ボードのブラケットのネジを取り外します。

(4)　取り外したネジは、スロットカバーの固定に使用します。
本ボードを、PCI バススロットから取り外します。

(5)　スロットカバーを、手順(3)で取り外したネジで固定します。

(6)　パソコンのカバーを取り付け、ケーブル類を元の状態にして下さい。

⚠ 注意

● ボードに衝撃を与えたり、曲げたりしないで下さい。誤動作、故障の原因になります。

本ボードには多数の電気部品が搭載されていますが、これらは静電破壊を起こす恐れがあります。
取扱い前には本ボード以外の金属部等に触れ、作業者自身の放電を行って下さい。又、取扱い時も電気部品には触れぬ様にご注意下さい。

● ボードの金メッキ端子部（エッジコネクタ）には、手を触れないで下さい。
誤動作、故障の原因となります。

● パソコンの電源が入った状態で、ボードを PCI バススロットに実装したり、抜いたりしないで下さい。
誤動作、故障の原因となります。必ずパソコンの電源を切ってから実施して下さい。

６．仕様

６-１． 基本仕様

センサ接続数(ch 数)		4 個
ケーブル長		CAT5 ケーブル、最大 30 m
使用温度湿度範囲	温度	0 ℃ to 50 ℃
	湿度	20 % to 90 %(結露無き事)
PCI バス仕様		32 bit/33 MHz、5 V(1.5 A)
外形寸法		106.68(H)×174.63(L)mm（パネル及び突起部含まず）
質量		約 133 g

⚠ 注意

・本ボードに十分な電力が供給される PCI 規格のパソコンをご使用下さい。

７．保証

７-１.保証

- 本ボードの保証期間は、本ボード納入後１年間です。
- 保証期間中の修理、アフターサービスは、購入された弊社営業所、又は代理店等にご相談下さい。

７-２.修理

修理を依頼される場合は、もう一度、接続、設定が確実に行われているか確認して下さい。
確認の結果、それでも異常があると認められた時は、本ボードを購入された弊社営業所、又は代理店に依頼して下さい。

※記載されている内容は、改良のため予告なく変更することがあります。

ミネベア株式会社

本　　社	〒389−0293 長野県北佐久郡御代田町大字御代田 4106−73	☎0267−32−2200 FAX.0267−31−1350
計測販売部関東	〒251−8531 神奈川県藤沢市片瀬1−1−1	☎0466−23−2180 FAX.0466−22−7191
計測販売部関西	〒541−0053 大阪市中央区本町1−7−7 (WAKITA堺筋本町ビル11F)	☎06−6263−8331 FAX.06−6263−7388
計測販売部名古屋	〒460−0003 名古屋市中央区錦1−6−5 (名古屋錦シティビル4F)	☎052−231−1181 FAX.052−231−1157

計測機器事業部

藤沢工場	〒251−8531神奈川県藤沢市片瀬1−1−1	☎0466−22−7151 FAX.0466−22−1701
軽井沢工場	〒389−0293長野県北佐久郡御代田町大字御代田 4106−73	☎0267−31−1309 FAX.0267−31−1353
	テクニカルサポートフリーダイヤル	0120−950008
	ホームページアドレス	http://www.minebea−mcd.com